Effective Representation of RT-LOTOS Terms by Finite Time Petri Nets

Tarek Sadani[1,2], Marc Boyer[3], Pierre de Saqui-Sannes[1,2], and Jean-Pierre Courtiat[1]
tsadani@ensica.fr, mboyer@enseeiht.fr, desaqui@ensica.fr, courtiat@laas.fr

[1] LAAS-CNRS, 7 av. du colonel Roche, 31077 Toulouse Cedex 04, France
[2] ENSICA, 1 place Emile Blouin, 31056 Toulouse Cedex 05, France
[3] IRIT-CNRS/ENSEEIHT, 2 rue Camichel, 31000 Toulouse, France

Abstract. The paper describes a transformational approach for the specification and formal verification of concurrent and real-time systems. At upper level, one system is specified using the timed process algebra RT-LOTOS. The output of the proposed transformation is a Time Petri net (TPN). The paper particularly shows how a TPN can be automatically constructed from an RT-LOTOS specification using a compositionally defined mapping. The proof of the translation consistency is sketched in the paper and developed in [1]. The RT-LOTOS to TPN translation patterns formalized in the paper are being implemented. in a prototype tool. This enables reusing TPNs verification techniques and tools for the profit of RT-LOTOS.

1 Introduction

The design of time-critical systems is a complex task. Given the risk to not detect transient errors by using conventional techniques such as simulation or testing, it is strongly recommended to use formal verification techniques such as model checking, that have been proven to facilitate early detection of design errors, and to contribute to produce systems at a correctness level that cannot be reached by using simulation and testing techniques.

The use of formal verification techniques is usually linked to the use of formal specifications. Among the wealth of formal specification techniques proposed in the literature, process algebras play a special role. Their compositional operators allow one to describe a system made up of components that communicate and operate concurrently. Besides its notion of compositionality, the capacity to model real-time mechanisms is an essential feature of RT-LOTOS [2, 3], the timed process algebra addressed in this paper.

Several verification tools have been developed for timed process algebras. Few of them are really efficient. They usually implement translations into timed automata, which permits to reuse model checkers such as [4, 5]. Petri nets verification tools may be considered as well. The possibility to reuse a Time Petri Nets

analyzer for verifying RT-LOTOS specifications is one of the main motivation behind the work presented in this paper.

The RT-LOTOS to TPN translation approach discussed in this paper relies on the *TPN component* model first introduced in [6]. The model published in [6] is extended and improved. Discussion is not restricted to RT-LOTOS. The paper highlight difficulties and most important issues one might face while translating timed process algebras into Time Petri nets.

The paper is organized as follows. Section 2 introduces the RT-LOTOS language. Section 3 introduces the Time Petri net (TPN) model. Section 4 details RT-LOTOS to TPN translation patterns and explains the intuition behind the proof. In particular, it is shown how TPNs are embedded in components and composed. Section 5 surveys related work. We particularly compare our approach with the Petri Box Calculus [7]. Section 6 concludes the paper.

2 RT-LOTOS

The Language of Temporal Ordering Specifications (LOTOS[8]) is a formal description technique based on CCS [9] and extended by a multi-way synchronization mechanism inherited from CSP [10]. RT-LOTOS [2] extends LOTOS with three temporal operators: a *deterministic delay*, a *latency operator* which enables description of temporal indeterminism and a *time limited offer*. The main difference between RT-LOTOS and other timed extensions of LOTOS lies in the way a non-deterministic delay may be expressed. RT-LOTOS supports the so-called *latency* operator. Its usefulness and efficiency have been proved in control command applications and hypermedia authoring [11].
The following processes `P` and `PL` illustrate the use of the three temporal operators of RT-LOTOS.

```
Process P[a]: exit:=
 delay(2)a{5}; exit
endproc
```

```
Process PL[a]: exit:=
 delay(2)latency(6)a{5}; exit
endproc
```

Process `P` starts with a 2 time units delay. Once the delay expires, action `a` is offered to the environment during 5 time units. If the process's environment does not synchronize on `a` before this deadline, a time violation occurs and the process transforms into `stop`. Process `PL` differs from `P`, for it contains a *latency* operator. Action `a` is delayed by a minimum dealy of 2 units of time and a maximum delay of 8 units of time(in case the *latency* goes to its maximum value). From the environment's point of view, if the latency lasts l time units, the process behaves like `delay(2+`l`)a{5-`l`}` (cf. the left part of Fig. 1). Of course, if the duration of the latency goes beyond 5 units of time, a temporal violation occurs and process `PL` transforms into `stop`(cf. the right part of Fig. 1).

The originality and interest of the latency operator is more obvious when one combines that operator with the *hiding* operator. In LOTOS, hiding allows one to transform an external *observable* action into an *internal* one. In RT-LOTOS, hiding has the form of a renaming operator which renames action `a` into `i(a)`.

In most timed extensions of LOTOS, hiding implies urgency. It thus removes any time indeterminism inherent to the limited time offering. In RT-LOTOS, a hidden action is urgent *as soon as it is no longer delayed by some latency operator*. Let us, e.g., consider the RT-LOTOS behavior `hide a in PL` where action `a` is hidden in process `PL`. If l is the duration of the latency, `i(a)` will *necessarily* occur at date $2 + l$, if $l < 5$. (cf. Fig. 2). But, if $(l > 5)$, a temporal violation occurs (similarly to the situation where action `a` was an observable action).

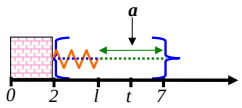
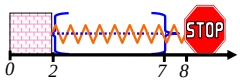


Fig. 1. Combining delay, latency and limited offering



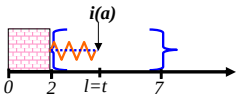


Fig. 2. Adding hiding

Let us now point out some differences between RT-LOTOS and E-LOTOS [12]. In E-LOTOS, urgency may apply to observable actions as soon as the latter are defined as exceptions. Conversely, the RT-LOTOS semantics states that one cannot enforce urgency on visible events. The only way to introduce urgency in RT-LOTOS is to use the *hide* operator (cf Fig. 2). E-LOTOS further allows one to introduce temporal non-determinism by combining the operator used for non deterministic variable assignment with the deterministic delay operator (applied on the same variable). It is clear that E-LOTOS implements a data-oriented approach for specifying temporal non determinism (for example: `var t: time in ?t := any time [1<t<4]; wait (t) endvar; P`). Conversely RT-LOTOS implements a control oriented approach. In RT-LOTOS, the temporal non deterministic variable is a particular variable introduced by a specific operator. E-LOTOS and RT-LOTOS also use different ways to combine a non deterministic delay with a time limited offer. As depicted in Fig 1, the RT-LOTOS semantics states that the *latency* and the *time limited offer* start simultaneously. This makes it possible to express temporal violations when $t < l$. Conversely, for the E-LOTOS counterpart, the constraint on offering one action inside a time interval will not be active before the non deterministic delay elapses.

3 Time Petri nets

To our knowledge, Petri nets were the first theoretical model augmented with time constraints [13, 14], and the support of the first reachability algorithms for timed system [15, 16].

The basic idea of time Petri nets (TPN [13, 14]) is to associate an interval $I_s(t) = [a, b]$ (static interval) with each transition t. A transition *can* be fired if it has continuously been enabled during at least a time units, and it *must* fire if continuous enabling time reaches b time units. That is to say, once a transition is enabled ($M\,[t\rangle$), a *firing interval* $I_f(t)$ is created with initial value $I_s(t)$. Time

passing decreases the bounds of the interval. The transition may be fired once the lower bound reaches 0 and has to be fired when the upper bound reaches 0 (unless it conflicts with another transition). Figure 3 is a first example. In

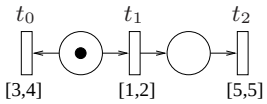


Fig. 3. Priority from urgency

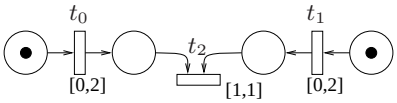


Fig. 4. Synchronization

the initial marking, only t_0 and t_1 are enabled. After one time unit delay, t_1 is firable. Because t_1 reaches its upper interval always before t_0 becomes enabled ($3 > 2$), then t_0 can never be fired. t_2 is fired five time units after the firing of t_1. Figure 4 illustrates the synchronization rule: t_0 (resp. t_1) is fired at an absolute date $\theta_0 \leq 2$ (resp. $\theta_1 \leq 2$), and t_2 is fired at $max(\theta_0, \theta_1) + 1$.

4 Translation from RT-LOTOS into TPN

The quality of a translation depends on its capability to guarantee a close relation between the properties that hold in the source and those that still hold in the target [17]. This is why we defined a one-to-one mapping of actions between RT-LOTOS and TPNs. Since we do not use auxiliary transitions, we ensure that the proposed translation patterns do not add any behavior. Moreover, RT-LOTOS is compositional by nature. It is then inevitable, during the translation procedure, to consider TPNs as composable entities. Unfortunately, TPNs in their original form miss a convenient way of composing or decomposing larger nets from or to smaller ones by means of a set of high level operators. We solve this problem by introducing the concept of *TPN component* as basic building block. We also define a set of operations on components (Section 4.2). These operations match the composition and temporal operators supported by RT-LOTOS.

4.1 Time Petri net Component

A *Component* encapsulates a labeled TPN which describes its behavior. A component is endowed with interfaces and interactions points. It performs an action by firing the appropriate transition. A component has two sets of labels: *Act* the alphabet of the component and $Time = \{\text{tv}, \text{delay}, \text{latency}\}$. These three labels are introduced to represent the temporal behavior of components. The tv (for "temporal violation") label represents a time-limited offer expiration. A delay or latency label represents the expiration of some deterministic or non deterministic delay, respectively.

A component is graphically represented by a box containing one TPN. The black-filled boxes at the component boundary represent interaction points. For

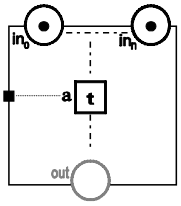


Fig. 5. Component example

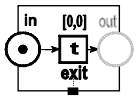


Fig. 6. The `exit` pattern

instance, the component $C_{\mathtt{P}}$ in Figure 5 is built from some RT-LOTOS term `P`. During its execution, it may perform observable action `a`. The in_i (initially marked places) represent the component input interface, and the *out* place denotes its output interface. A token in the *out* place of a component means that the component has successfully completed its execution. A component is *activated* by filling its input places. A component is *active* if at least one of its transitions is enabled. Otherwise, the component is *inactive*.

Definition 1 (Component).

Let $Act = A_o \cup A_h \cup \{exit\}$ be an alphabet of actions, where A_o is a set of observable actions *(with $i \notin A_o$, $exit \notin A_o$), $A_h = \{i\} \times A_o$ is the set of hidden actions (If $\mathtt{a}$ is an observable action, $i_{\mathtt{a}}$ denotes a hidden action).*

A component is a tuple $C = \langle \Sigma, Lab, I, O \rangle$ where

- *$\Sigma = \langle P, T, Pre, Post, M_0, IS \rangle$ is a TPN.*
- *$Lab : T \rightarrow (Act \cup Time)$ is a labeling function which labels each transition in Σ with either an action name (Act) or a time-event ($Time = \{tv, delay, latency\}$). Let T^{Act} (resp. T^{Time}) be the set of transitions with labels in Act (resp. $Time$).*
- *$I \subset P$ is a non empty set of places defining the input interface.*
- *$O \subset P$ is the output interface of the component. A component has an output interface if it has at least one transition labeled by `exit`. If so, O is the outgoing place of those transitions. Otherwise, $O = \emptyset$.*

Moreover, a set of invariants is associated with the components:

H1 There is no source transition in a component.

H2 The encapsulated TPN is 1-bounded (cf. *safe* nets in [7]). H2 is called the "safe marking" property. It is essential for the decidability of reachability analysis procedure applied to TPNs.

H3 If all the input places are marked, all other places are empty ($I \subset M \Rightarrow M = I$).

H4 If the *out* place is marked, all other places are empty ($O \neq \emptyset \wedge O \subset M \Rightarrow M = O$).

H5 For each transition t such that $Lab(t) \in Act$, if the label is an observable action ($Lab(t) \in A_0$), its time interval is $[0, \infty)$, otherwise[4], it is $[0, 0]$.

Hypotheses H3–H4 are called *clean markings* in [7].

[4] $Lab(t) \in A_h \cup \{exit\}$

4.2 Translation patterns

When translating RT-LOTOS specifications into TPNs, we associate a specific operation (involving some component(s)) with each RT-LOTOS operator. These operations are graphically depicted through a set of patterns presented in next sections. To these graphical translation patterns, we add a complementary formal definition. For space reasons, the formalization of some patterns is not presented in this paper. A complete formal definition can be found in the extended version of this paper [1].

Notation: $f' = f \cup (a, b)$ denotes the function $f' : A \cup \{a\} \mapsto B \cup \{b\}$ such that $f'(x) = f(x)$ if $x \in A$ and $f'(a) = b$ otherwise .

Low level Petri net operations. The formal definition of the translation patterns uses the following low level Petri nets operators: $\cup, \setminus, \uplus$.
Let $N = \langle P, T, Pre, Post, M_0, IS\rangle$ be a TPN.
Adding a place: Let p be a new place ($p \notin P$), Pre_p and $Post_p$ two sets of transitions of T. $N' = N \cup \langle Pre_p, p, Post_p\rangle$ is the TPN augmented with place p such that ${}^\bullet p = Pre_p$ and $p^\bullet = Post_p$.

$$N' = \langle P \cup \{p\}, T, Pre \cup \bigcup_{t \in Pre_p} (p, t), Post \cup \bigcup_{t \in Post_p} (t, p), M_0, IS\rangle$$

Adding a transition: Let t be a new transition ($t \notin T$), and I its time interval, Pre_t and $Post_t$ two sets of places of P. $N' = N \cup \langle Pre_t, (t, I), Post_t\rangle$ is the TPN augmented with transition t such that ${}^\bullet t = Pre_t$ and $t^\bullet = Post_t$.

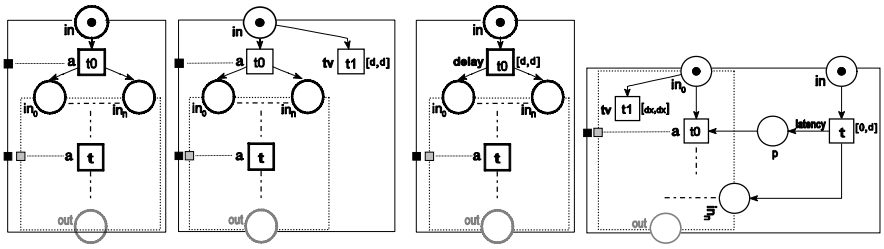
$$N' = \langle P, T \cup \{t\}, Pre \cup \bigcup_{p \in Pre_t} (p, t), Post \cup \bigcup_{p \in Post_t} (t, p), M_0, IS \cup (t, I)\rangle$$

Basic components The $C_{\texttt{stop}}$ component is simply the empty net (no place, no transition). The $C_{\texttt{exit}}$ is a component which performs a successful termination. It has one input place, one output place, and a single transition labelled with `exit` and a static interval $[0, 0]$ (Fig.6).

Patterns applying to one component Let us consider the component $C_{\texttt{P}}$ of Fig. 5. Fig. 7 depicts different patterns applied to $C_{\texttt{P}}$.

- $C_{\texttt{a;P}}$ (Fig. 7(a)) is the component resulting from prefixing $C_{\texttt{P}}$ with action `a`. $C_{\texttt{a;P}}$ executes `a` then activates $C_{\texttt{P}}$.
  $C_{\texttt{a;P}} = \langle \Sigma_{\texttt{a;P}}, Lab_{\texttt{a;P}}, \{in\}, O_{\texttt{P}}\rangle$ where the TPN $\Sigma_{\texttt{a;P}}$ is obtained by adding a place in and a transition t_0 to $\Sigma_{\texttt{P}}$, $Lab_{\texttt{a;P}}$ associates `a` to transition t_0.

$$\Sigma_{\texttt{a;P}} = (\Sigma_{\texttt{P}} \cup \langle \emptyset, (t_0, [0, \infty)), I_{\texttt{P}}\rangle) \cup \langle \emptyset, in, t_0\rangle$$
$$Lab_{\texttt{a;P}} = Lab_{\texttt{P}} \cup (t_0, \texttt{a})$$



(a) a;P (b) a{d}P (c) delay(d)P (d) latency(d)P

Fig. 7. Patterns applying to one component

– $C_{\texttt{a\{d\};P}}$ (Fig. 7(b)) is the component resulting from prefixing $C_{\texttt{P}}$ with a limited offer of d units of time on action a. If for any reason, a cannot occur during this time interval, the tv transition will be fired (temporal violation situation) and $C_{\texttt{a\{d\};P}}$ will transform into an inactive component.
The pattern is very similar to the one of $C_{\texttt{a;P}}$. Therefore, its definition reuses that of $C_{\texttt{a;P}}$.

$$C_{\texttt{a\{d\};P}} = \langle \Sigma_{\texttt{a\{d\};P}}, Lab_{\texttt{a;P}} \cup \{(t_1, \text{tv})\}, \{in\}, O_{\texttt{P}} \rangle$$
$$\Sigma_{\texttt{a\{d\};P}} = \Sigma_{\texttt{a;P}} \cup \langle \{in\}, (t_1, [d, d]), \emptyset \rangle$$

– $C_{\texttt{delay(d)P}}$ (Fig 7(c)) is the component resulting from delaying the first action of P with a deterministic delay of d units of time. This is exactly the same pattern as $C_{\texttt{a;P}}$ except that the added transition has a delay label and a static interval equal to $[d, d]$.

$$C_{\texttt{delay(d)P}} = \langle \Sigma_{\texttt{delay(d)P}}, Lab_{\texttt{P}} \cup \{(t_0, \text{delay})\}, \{in\}, O_{\texttt{P}} \rangle$$
$$\Sigma_{\texttt{delay(d)P}} = (\Sigma_{\texttt{P}} \cup \langle \emptyset, (t_0, [d, d])), I_{\texttt{P}} \rangle) \cup \langle \emptyset, in, t_0 \rangle$$

– $C_{\texttt{latency(d)P}}$ (Fig 7(d)) is the component resulting from delaying the first actions of $C_{\texttt{P}}$ with a non deterministic delay of d units of time.
Like the delay operator, the latency operator is defined by connecting a new transition to the input interface of $C_{\texttt{P}}$. This time, we add a static interval equal to $[0, d]$. The definition of the latency translation pattern must cope with the "subtle" situation where one (or several) action(s) among $C_{\texttt{P}}$'s first actions is (are) constrained with a limited offer (this set is denoted by $\mathcal{FA}_{lo}$). For instance, in Fig 7(d), action a is offered to the environment during d_x units of time. The RT-LOTOS semantics states that the latency and the offering of a start simultaneously, which means that if the latency duration goes beyond d_x units of time, the offer on a will expire. To obtain the same behavior, we add the input place in_0 of a to the input interface of the resulting component $C_{\texttt{latency(d)P}}$. In the definition of the pattern, we denote I_{lo} the set of these input places ($I_{lo} \subset I_{\texttt{P}}$). Thus t_1 and t are enabled as soon as the component is activated (all its input places being marked). $C_{\texttt{latency(d)P}}$ is able to execute a (fire t_0) if t_0 is enabled (i.e if in_0 and p are marked) before

t_1 is fired (at d_x). Therefore, action a is possibly offered to the environment for no more than d_x units of time, hence conforming to the RT-LOTOS semantics.
Let $\mathcal{FA}(C_{\mathtt{P}})$ be the set of transitions associated to the first actions of P[5], and $\mathcal{FA}_{lo}(C_{\mathtt{P}})$ be the set of first actions constrained by a time limited offer:

$$\begin{aligned}
\mathcal{FA}_{lo}(C_{\mathtt{P}}) &= \left\{ t_a \in \mathcal{FA}(C_{\mathtt{P}}) \;\middle|\; \mathrm{tv} \in ({}^{\bullet}t_a)^{\bullet} \right\} \\
I_{lo} &= {}^{\bullet}\mathcal{FA}_{lo}(C_{\mathtt{P}}) \\
C_{\mathtt{latency(d)P}} &= \langle \Sigma_{\mathtt{latency(d)P}}, Lab_{\mathtt{P}} \cup \{(t, \mathrm{latency})\}, I_{lo} \cup \{in\}, O_{\mathtt{P}} \rangle \\
\Sigma_{\mathtt{latency(d)P}} &= \Sigma_{\mathtt{P}} \cup \bigcup_{t_a \in \mathcal{FA}_{lo}(C_{\mathtt{P}})} \langle t, p_{t_a}, t_a \rangle \cup \langle \emptyset, in, \emptyset \rangle \\
&\quad \cup \left\langle \{in\}, (t, [0, d]), (I_{\mathtt{P}} \backslash I_{lo}) \cup \bigcup_{t_a \in \mathcal{FA}_{lo}(C_{\mathtt{P}})} \{p_{t_a}\} \right\rangle
\end{aligned}$$

- $C_{\mu \mathtt{X.(P;X)}}$ is the component which executes $C_{\mathtt{P}}$'s actions ad infinitum. The recursion operator translation is mainly an untimed problem. It is not presented in this paper, since the focus is laid on timed aspects.
- $C_{\mathtt{hide\ a\ in\ P}}$ is the component resulting from hiding action a in $C_{\mathtt{P}}$. Hiding allows one to transform observable (external) actions into unobservable (internal) actions, then making the latter unavailable for synchronization with other components. In RT-LOTOS, hiding one or several actions induces a notion of urgency on action occurrence. Consequently, a TPN transition corresponding to one hidden action will be constrained by a time interval equal to $[0, 0]$. This implies that as soon as a transition is enabled, it is candidate for being fired.

Patterns applying to a set of components Each of the following patterns transforms a set of components into one component.

- $C_{\mathtt{P|[a]|Q}}$ (Fig.8)
  The concept of handshake communication is an important feature of process algebras. It consists of a symmetric synchronization by which an action that is shared between n processes can be executed only if all of them are ready to do so. In Petri nets, such a scenario is represented by a transition with n input places. This transition can fire only if all its input places contain a token (cf. Fig. 4). At the PN level, the synchronization operation is achieved through transition merging. While transitions merging is straightforward in Petri nets, it turns to be a rather tricky issue in Time Petri nets. Indeed, it requires explicit handling of the time intervals assigned to transitions to be merged. These time intervals may be incompatible, which leads to express global timing constraints as a conjunction of intervals whose consistency is not guaranteed. This problem is not solved in [18](where each transition is assigned a time interval), as presented in Sect. 5.2.

[5] Its formal definition is given in Def. 2, Sect. A.

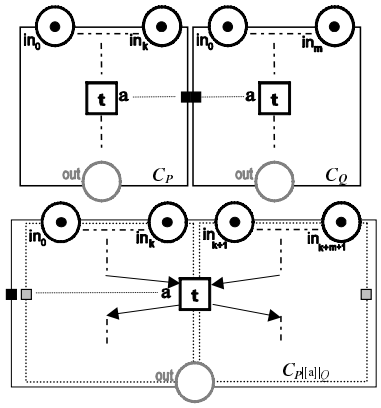


Fig. 8. Parallel synchronization pattern

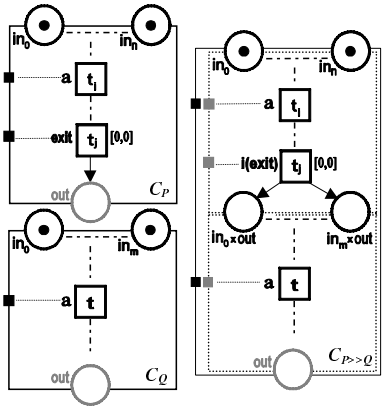


Fig. 9. Sequential composition pattern

To solve this problem and make transition merging always a possible operation, we avoid assigning time intervals to action transitions. Instead, the timing constraints are assigned to dedicated transitions (cf. time limited offer pattern).

The synchronization on `a` of $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$ is achieved by merging each `a` transition in $C_{\mathtt{P}}$ with each `a` transition in $C_{\mathtt{Q}}$, thus creating $n * m$ `a` transitions in $C_{\mathtt{P|[a]|Q}}$ (n and m being the number of `a` transitions in $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$, respectively).

– $C_{\mathtt{P>>Q}}$ (Fig. 9) depicts a sequential composition of $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$ which means that if $C_{\mathtt{P}}$ successfully completes its execution then it activates $C_{\mathtt{Q}}$. This kind of composition is possible only if $C_{\mathtt{P}}$ has an output interface. The resulting component $C_{\mathtt{P>>Q}}$ is obtained by merging the output interface of $C_{\mathtt{P}}$ and the input interface of $C_{\mathtt{Q}}$, and by hiding the `exit` interaction point of $C_{\mathtt{P}}$.

$$C_{\mathtt{P>>Q}} = \langle \Sigma_{\mathtt{P>>Q}}, Lab_{\mathtt{hide\ exit\ in\ P}} \cup Lab_{\mathtt{Q}}, I_{\mathtt{P}}, 0_{\mathtt{Q}} \rangle$$
$$\Sigma_{\mathtt{P>>Q}} = \langle P_{\mathtt{P}} \backslash O_{\mathtt{P}} \cup P_{\mathtt{Q}}, T_{\mathtt{hide\ exit\ in\ P}} \cup T_{\mathtt{Q}}, Pre_{\mathtt{P}} \cup Pre_{\mathtt{Q}}, Post_{\mathtt{P>>Q}}, IS_{\mathtt{P}} \cup IS_{\mathtt{Q}} \rangle$$
$$Post_{\mathtt{P>>Q}} = (Post_{\mathtt{P}} \backslash \{(t, O_{\mathtt{P}}) \mid t \in {}^{\bullet}O_{\mathtt{P}}\}) \cup \{(t, in_{\mathtt{Q}}) \mid in_{\mathtt{Q}} \in I_{\mathtt{Q}} \wedge t \in {}^{\bullet}O_{\mathtt{P}}\} \cup Post_{\mathtt{Q}}$$

– $C_{\mathtt{P[]Q}}$ (Fig. 10) is the component which behaves either as $C_{\mathtt{P}}$ or $C_{\mathtt{Q}}$.
We do not specify whether the choice between the alternatives is made by the component $C_{\mathtt{P[]Q}}$ itself, or by the environment, but it should be made at the level of the first actions in the component. In other words, the occurrence of one of the first actions in either component determines which component will continue its execution and which one must be deactivated. The problem can be viewed as a competition between $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$. These two components compete to execute their first action. As long as the latter has not yet occurred, $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$ age similarly, which means that $Time$ transitions (labeled by tv, delay or latency) may occur in both components without any consequence on the choice of the wining component. Once one first action occurs, the control is irreversibly transferred to the winning component. The other

one is deactivated, in the sense that it no longer contains enabled transitions. The *choice* operator is known to cause trouble in presence of initial parallelism. [19] defines a choice operator where each alternative has just one initial place. Therefore, none of the alternative allows any initial parallelism. We think that it is a strong restriction. We do not impose any constraint on the *choice* alternatives.

The solution we propose to define a choice between two components is as follows: to obtain the intended behavior, we introduce a set of special places, called *lock* places. Those places belong to the input interface of component $C_{\mathtt{P[]Q}}$. Their function is to undertake control transfer between the two components. For each first action of $C_{\mathtt{P}}$ we introduce one lock place per concurrent first action in $C_{\mathtt{Q}}$ (for instance `a` has one concurrent action in $C_{\mathtt{Q}}$: `c`, while `c` has two concurrent actions in $C_{\mathtt{P}}$: `a` and `b`) and vice versa. A *lock* place interacts only with those transitions representing the set of initial actions and the time labeled transitions they are related with (delay for `a` and tv for `b`). *Time* transitions restore the token in the *lock* place, since they do not represent an action occurrence, but a time progression which has not to interfere with the execution of the other component (as long as the first action has not occurred, the two components age similarly). The occurrence of an initial action of $C_{\mathtt{P}}$ (respectively $C_{\mathtt{Q}}$) locks the execution of $C_{\mathtt{Q}}$ (respectively $C_{\mathtt{P}}$) by stealing the token from the *lock* places related to all $C_{\mathtt{Q}}$'s (respectively $C_{\mathtt{P}}$'s) first actions. A unique *out* place is created by merging the *out* places of $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$.

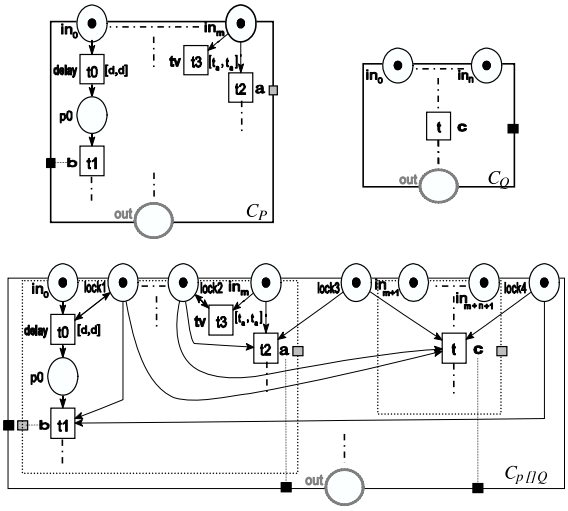


Fig. 10. Choice between $C_{\mathtt{P}}$ and $C_{\mathtt{Q}}$

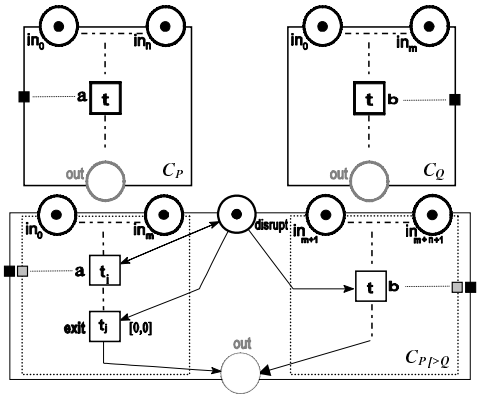


Fig. 11. The disrupt pattern

– $C_{\mathtt{P[>Q}}$ (Fig. 11) is the component representing the behavior where component $C_{\mathtt{P}}$ can be interrupted by $C_{\mathtt{Q}}$ at any time during its execution. It means that

at any point during the execution of $C_{\mathtt{P}}$, there is a choice between executing one of the next actions from $C_{\mathtt{P}}$ or one of the first actions from $C_{\mathtt{Q}}$. For this purpose, $C_{\mathtt{Q}}$ *steals* the token from the shared place named `disrupt` (which belongs to the input interface of $C_{\mathtt{P}[>\mathtt{Q}}$). Thus the control is irreversibly transferred from $C_{\mathtt{P}}$ to $C_{\mathtt{Q}}$ (`disrupt` is an *input* place for $C_{\mathtt{Q}}$ first action and `exit` transition of $C_{\mathtt{P}}$; it is also an *input/output* place for all the others transitions of $C_{\mathtt{P}}$). Once an action from $C_{\mathtt{Q}}$ is chosen, $C_{\mathtt{Q}}$ continues executing, and $C_{\mathtt{P}}$'s transitions are no longer enabled.

4.3 Sketch of the proof

We prove that the translation preserves the RT-LOTOS semantics and that the defined compositional framework preserves the good properties (H1–H5) of the components.

Intuitively an RT-LOTOS term and a component are timed bisimilar [20] iff they perform the same action at the same time and reach bisimilar states. For each operator, we prove that, from each reachable state, if the occurrence of a time progression (respectively an action) is possible in an RT-LOTOS term, it is also possible in its associated component, and conversely. Therefore, we ensure that the translation preserves the sequences of possible actions but also the occurrence dates of these actions. The entire proof may be found in [1].

5 Related work

Much work has been done on translating process algebras into Petri Nets, by giving a Petri net semantics to process terms [21, 19, 22]. [22] suggests that a good net semantics should satisfy the retrievability principle, meaning that no new "auxiliary" transitions should be introduced in the reachability graph of the Petri net. [21, 19] do not satisfy this criterion. In this paper, we define a one-to-one mapping which is compliant with this strong recommendation.

5.1 Untimed models

A survey of the literature indicates that proposals for LOTOS to Petri net translations essentially address the untimed version of LOTOS [23–28]. The opposite translation has been discussed by [27] where only a subset of LOTOS is considered, and by [29] where the authors addressed the translation of Petri nets with inhibitor arcs into basic LOTOS by mapping places and transitions into LOTOS expressions. [26] demonstrated the possibility to verify LOTOS specifications using verification techniques developed for Petri nets by implementing a Karp and Miller procedure in the LOTOS world.

[23, 28] operate a complete translation of LOTOS, handling both the control and data parts. Moreover, they just consider regular LOTOS terms. So do we. The LOTOS to PN translation algorithms of [23, 28] were implemented in the CAESAR tool. Besides the temporal aspects addressed in this paper, a technical

difference with [23, 28] lies in the way we structure TPNs. Our solution is based on TPNs components. In our approach, a component may contain several tokens. Conversely, [23, 28] structure Petri nets into units, each of them containing one token at most. This invariant limits the size of markings, and permits optimizations on memory consumption. The counterpart is that [23, 28] use ϵ-transitions. The latter introduce non determinism. They are eliminated when the underlying automaton is generated (by transitive closure). The use of ϵ-transitions may be inefficient in some particular cases (see the example provided in [6])

The major theoretical study on taking advantage of both Petri nets and process algebras is presented in [7]. The proposed solution is Petri Box Calculus (PBC), a generic model that embodies both process algebra and Petri nets. The authors start from Petri nets to come up with a CCS-like process algebra whose operators may straightforwardly be expressed by means of Petri nets.

5.2 Timed models

[30] pioneered work on timed enhancements of the control part of LOTOS inspired by timed Petri nets models. [31] defined a mapping from TPNs to TE-LOTOS which makes it possible to incorporate basic blocks specified as 1-bounded TPNs into TE-LOTOS specifications. However, because of the strong time semantics of TPNs (a transition is fired as soon as the upper bound of its time interval is reached unless it conflicts with another one) a direct mapping was not always possible.

Timed extensions of PBC have been proposed in [18, 32]. Although the component model proposed in this paper is not a specification model but an intermediate model used as gateway between RT-LOTOS and TPNs, we find it important to compare our work with [18].

Of prime interest to us is the way [18] introduces temporal constraints in his framework by providing each action with two time bounds representing the earliest firing time and latest firing time. This approach is directly inspired by TPNs, where the firing of actions is driven by necessity. However, a well known issue with this strategy is that it is inappropriate for a compositional and incremental building of specifications. The main difficulty is to compose time intervals when dealing with actions synchronization. The operational semantics of [18] relies on intervals intersection to calculate a unique time interval for a synchronized transition. However, this approach is not always satisfactory, as shown in the following example.

let us consider the following timed PBC term:
$E_1 = ((a[10, 10]; b[2, 2]) \mid\mid \hat{b}[12, 12])\ sy\{b\}$. It expresses the parallel synchronization on b (The synchronization of two conjugate actions b and $\hat{b}$ gives rise to the silent action i) of the following terms:

- $a[10, 10]; b[2, 2]$ executes a after 10 time moves (time has a discrete semantics), then it executes b, 2 time moves after the occurrence of a,
- $\hat{b}[12, 12]$ executes $\hat{b}$ after 12 times moves.

That is to say, $\hat{b}$ and b occur at the same date (12 times units after the initial instant). Thus the synchronization on b should be possible. Nevertheless E_1 cannot execute the synchronization action i since $[2,2] \cap [12,12] = \emptyset$. The synchronization rule of [18] states that the synchronization is possible only if it leads to a *well-defined* action, i.e. with consistent timing information. Therefore, the corresponding TPN (called *ctbox* in [18]) cannot be constructed.

To avoid this difficult interval composition, we do not assign any time interval to action transitions. In our framework, timed constraints are assigned to dedicated transitions (cf Fig. 7(b), 7(c) and 7(d)). Thus action transitions are free from any timing constraint. This way, the synchronization can be obtained by merging the action transitions without changing the timing constraints. Hence, we are able to straightforwardly construct the TPN of Fig. 12 (obtained by the application of the pattern of Fig 7(c), 7(a) and 8) corresponding to the following RT-LOTOS expression which is behaviorally equivalent to the above timed PBC expression:

```
P1= Hide a ,b in (     (delay(12)b; stop)
                 |[b]| (delay(10)a; delay(2)b; stop))
```

Synchronization on b occurs indeed 12 time units after initial instant.

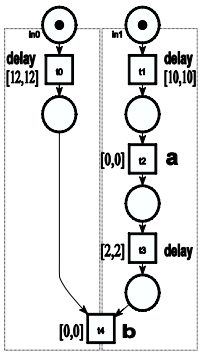


Fig. 12. C_{P1}

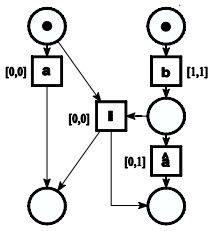


Fig. 13. ctbox$_1$

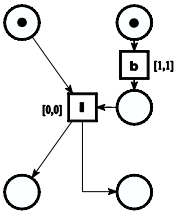


Fig. 14. ctbox$_2$

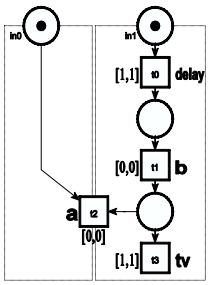


Fig. 15. C_{P2}

Moreover, even if the synchronization leads to a *well-defined* action. It was necessary to the author of [18] to enrich the semantics of the timed PBC with rules for allowing the firing of *illegal actions*. An action is said *illegal* if it has inconsistent timing information. For example, an action which has been enabled for an amount of time which exceeds its latest bound firing time.
As an illustration, let us now consider the following timed PBC expression taken from [18]: $E_2 = (a[0,0]||(b[1,1];\hat{a}[0,1]))sy\{a\}$. E_2 cannot execute the synchronization action i which is consistent with the TPN semantics (cf Fig 13). However, the situation changes for $E_2 rs\{a\}$. The synchronization is enforced by the restriction operator. The corresponding net is obtained by removing a and $\hat{a}$

transitions (cf Fig 14). Now it is possible to execute b at $[1,1]$ followed immediately by i. However, rules based only on legal actions could not produce a similar result in the operational semantics since no legal action occurrence is possible for $a[0,0]$ after the elapsing of 1 time unit. Therefore, a rule permitting the firing of a after one unit of time was introduced. The rules on allowing *illegal actions* were certainly unavoidable to ensure behavioural consistency between Timed PBC terms and their translation into Time Petri nets, but they badly impact on the simplicity of the operational semantics of [18].
The following RT-LOTOS expression which is behaviourally equivalent to $E_2rs\{a\}$ `: P2= Hide a,b in (a;stop)|[a]|(delay(1)b;a{1};stop)` has to our opinion a more intuitive behavior. The consistency between the RT-LOTOS term and its corresponding TPN of Fig 15 is ensured without departing from the original RT-LOTOS semantics.

Finally, [32] extends PBC with actions durations. This model captures timing information in a different manner. In our framework, actions are taken to be indivisible. As a consequence it is difficult to compare [32] with the approach proposed in this paper.

6 Conclusions

The paper discusses an efficient transformational approach for the verification of RT-LOTOS specifications. Our intent is to use the reachability graph of a TPN to represent and analyze the behavior of real time systems described in RT-LOTOS. After taking a closer look at the semantics of the two timed models, we formally define the concept of TPN component, together with a set of a translation patterns which match the set of RT-LOTOS operators. These patterns are implemented in RTL2TPN, a prototype tool which takes as input an RT-LOTOS specification and generates a TPN in a suitable format. Thus, it becomes possible to integrate TINA[33], a powerful TPN analyzer tool, to our verification platform. First experimental results are promising and confirm the advantage of using TPNs as an intermediate model [6]. It worth to be noticed that the transformation gives RT-LOTOS a formal semantics in terms of TPNs. Thus, we start with a top level RT-LOTOS view, which describes the architecture of the system together with its desired communication behavior. We end up with a more detailed view at the TPNs level, which describes the operational machine behavior of the system and clarifies the use of some RT-LOTOS operators, such as the *latency* operator.

This work is not limited to the verification of real-time systems specified in RT-LOTOS. The ultimate goal is to provide a more powerful verification environment for real-time systems modeled in TURTLE [34], a real-time UML profile whose formal semantics is expressed in RT-LOTOS.The translation patterns may be reused for other timed extensions of LOTOS, in particular ET-LOTOS.

References

1. Sadani, T., Boyer, M., de Saqui-Sannes, P., Courtiat, J.P.: Effective representation of regular RT-LOTOS terms by finite time petri nets. Technical Report 05605, LAAS/CNRS (2006)
2. Courtiat, J.P., Santos, C., Lohr, C., Outtaj, B.: Experience with RT-LOTOS, a temporal extension of the LOTOS formal description technique. Computer Communications **23**(12) (2000)
3. RT-LOTOS: Real-time LOTOS home page. (http://www.laas.fr/RT-LOTOS/)
4. Yovine, S.: Kronos: A verification tool for real-time systems. Software Tools for Technology Transfer **1**(123–133) (1997)
5. Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: Uppaal - a tool suite for automatic verification of real-time systems. In: Proc of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems. Number 1066 in LNCS (1995) 232–243
6. Sadani, T., Courtiat, J., de Saqui-Sannes, P.: From RT-LOTOS to time Petri nets. new foundations for a verification platform. In: Proc. of 3rd IEEE Int Conf on Software Engineering and Formal Methods (SEFM). (2005)
7. Best, E., Devillers, R., Koutny, M.: Petri Net Algebra. Monographs in Theoretical Computer Science: An EATCS Series. Springer-Verlag (2001) ISBN: 3-540-67398-9.
8. ISO - Information processing systems - Open Systems Interconnection: LOTOS - a formal description technique based on the temporal ordering of observational behaviour. ISO International Standard 8807:1989, ISO (1989)
9. Milner, R.: Communications and Concurrency. Prentice Hall (1989)
10. Hoare, C.: Communicating Sequential Processes. Prentice-Hall (1985)
11. Courtiat, J.P.: Formal design of interactive multimedia documents. In H.Konig, M.Heiner, A., ed.: Proc. of 23rd IFIP WG 6.1 Int Conf on Formal Techniques for Networked and distributed systems (FORTE'2003). Volume 2767 of LNCS. (2003)
12. ISO/IEC: Information technology - enhancements to LOTOS (E-LOTOS). Technical Report 15437:2001, ISO/IEC (2001)
13. Merlin, P.: A study of the recoverability of computer system. PhD thesis, Dep. Comput. Sci., Univ. California, Irvine (1974)
14. Merlin, P., Faber, D.J.: Recoverability of communication protocols. IEEE Transactions on Communications **COM-24**(9) (1976)
15. Berthomieu, B., Menasche, M.: Une approche par énumération pour l'analyse des réseaux de Petri temporels. In: Actes de la conférence IFIP'83. (1983) 71–77
16. Berthomieu, B., Diaz, M.: Modeling and verification of time dependant systems using Time Petri Nets. IEEE Transactions on Software Engineering **17**(3) (1991)
17. Katz, S., Grumberg, O.: A framework for translating models and specifications. In: Proc. of the 3d Int. Conf. on Integrated Formal Methods. (Volume 2335 of LNCS.)
18. Koutny, M.: A compositional model of time Petri nets. In: Proc. of the 21st Int. Conf. on Application and Theory of Petri Nets (ICATPN 2000). Number 1825 in LNCS, Aarhus, Denmark, Springer-Verlag (2000) 303–322
19. Taubner, D.: Finite Representations of CCS and TCSP Programs by Automata and Petri Nets. Number 369 in LNCS. Springer-Verlag (1989)
20. Yi, W.: Real-time behaviour of asynchronous agents. In: Proc. of Int. Conf on Theories of Concurrency: Unification and Extension (CONCUR). Volume 458 of LNCS. (1990)
21. Goltz, U.: On representing CCS programs by finite Petri nets. In: Proc. of Int. Conf. on Math. Foundations of Computer Science. Volume 324 of LNCS. (1988)

22. Olderog, E.R.: Nets, Terms, and formulas. Cambridge University Press (1991)
23. Garavel, H., Sifakis, J.: Compilation and verification of LOTOS specifications. In Logrippo, L., et al., eds.: Protocol Specification, Testing and Verification, X. Proceedings of the IFIP WG 6.1 Tenth International Symposium, 1990, Ottawa, Ont., Canada, Amsterdam, Netherlands, North-Holland (1990) 379–394
24. Barbeau, M., von Bochmann, G.: Verification of LOTOS specifications: A Petri net based approach. In: Proc. of Canadian Conf. on Electrical and Computer Engineering. (1990)
25. Larrabeiti, D., Quelmada, J., Pavón, S.: From LOTOS to Petri nets through expansion. In Gotzhein, R., Bredereke, J., eds.: Proc. of Int. Conf. on Formal Description Techniques and Theory, application and tools (FORTE/PSV'96). (1996)
26. Barbeau, M., von Bochmann, G.: Extension of the Karp and Miller procedure to LOTOS specifications. Discrete Mathematics and Theoretical Computer Science **3** (1991) 103–119
27. Barbeau, M., von Bochmann, G.: A subset of LOTOS with the computational power of place/transition-nets. In: Proc. of the 14th Int. Conf. on Application and Theory of Petri Nets (ICATPN). Volume 691 of LNCS. (1993)
28. Garavel, H., Lang, F., Mateescu, R.: An overview of cadp 2001. European Association for software science and technology (EASST) Newsletter **4** (2002)
29. Sisto, R., Valenzano, A.: Mapping Petri nets with inhibitor arcs onto basic LOTOS behavior expressions. IEEE Transactions on computers **44**(12) (1995) 1361–1370
30. Bolognesi, T., Lucidi, F., Trigila, S.: From timed Petri nets to timed LOTOS. In: Protocol Specification, Testing and Verification X (PSTV), Proceedings of the IFIP WG6.1 Tenth International Symposium on Protocol. (1990) 395–408
31. Durante, L., Sisto, R., Valenzano, A.: Integration of time Petri net and TE-LOTOS in the design and evaluation of factory communication systems. In: Proc. of the 2nd IEEE Workshop on Factory Communications Systems (WFCS'97). (1997)
32. Marroquin Alonso, O., de Frutos Escrig, D.: Extending the Petri box calculus with time,. In: Proc. of the 22nd International Conference on Application and Theory of Petri Nets (ICATPN). Volume 2075 of LNCS. (2001)
33. Berthomieu, B., Ribet, P., Vernadat, F.: The TINA tool: Construction of abstract state space for Petri nets and time Petri nets. Int. Journal of Production Research **42**(14) (2004)
34. Apvrille, L., Courtiat, J.P., Lohr, C., de Saqui-Sannes, P.: TURTLE : A real-time UML profile supported by a formal validation toolkit. IEEE Transactions on Software Engineering **30**(4) (2004)

A First actions

Definition 2 (First actions set). *Let C be a component. The set of first actions $\mathcal{FA}(C_P)$ can be recursively built using the following rules*[6]*:*

$$\mathcal{FA}(C_{stop}) = \emptyset \qquad \mathcal{FA}(C_{exit}) = \{t_{exit}\} \qquad \mathcal{FA}(C_{a;P}) = \mathcal{FA}\left(C_{a\{d\}P}\right) = \{t_a\}$$
$$\mathcal{FA}(C_{\mu X.(P;X)}) = \mathcal{FA}(C_{delay(d)P}) = \mathcal{FA}(C_{latency(d)P}) = \mathcal{FA}(C_{P;Q}) = \mathcal{FA}(C_{P>>Q}) = \mathcal{FA}(C_P)$$
$$\mathcal{FA}(C_{P|[A]|Q}) = \mathcal{FA}(C_{P[]Q}) = \mathcal{FA}(C_{P[>Q}) = \mathcal{FA}(C_P) \cup \mathcal{FA}(C_Q)$$
$$\mathcal{FA}(C_{hide\ a\ in\ P}) = h_a(\mathcal{FA}(C_P))$$

[6] where $t_{\mathtt{a}}$ is transition labelled by $\mathtt{a}$. $h_a(\alpha) = \alpha$ if $\alpha \neq a$ and $h_a(a) = i_a$